Noname manuscript No.
(will be inserted by the editor)

CURIE: A Cellular Automaton for Concept Drift Detection

Jesus L. Lobo · Javier Del Ser · Eneko Osaba · Albert Bifet · Francisco Herrera

Dedicated to Tom Fawcett and J.H.Conway, who passed away in 2020, for their noted contributions to the field of cellular automata and machine learning, and for inspiring this research.

Received: 09-19-2020 / Accepted: date

Abstract Data stream mining extracts information from large quantities of data flowing fast and continuously (data streams). They are usually affected by changes in the data distribution, giving rise to a phenomenon referred to as *concept drift*. Thus, learning models must detect and adapt to such changes, so as to exhibit a good predictive performance after a drift has occurred. In this regard, the development of effective drift detection algorithms becomes a key factor in data stream mining. In this work we propose *CURIE*, a drift detector relying on cellular automata. Specifically, in *CURIE* the distribution of the data stream is represented in the grid of a cellular automata, whose neighborhood rule can then be utilized to detect possible distribution changes over the stream. Computer simulations are presented and discussed to show that *CURIE*, when hybridized with other base learners, renders a competitive behavior in terms of detection metrics and classification accuracy. *CURIE* is compared with well-established drift detectors over synthetic datasets with varying drift characteristics.

Jesus L. Lobo
TECNALIA, Basque Research and Technology Alliance (BRTA),
E-mail: jesus.lopez@tecnalia.com

Javier Del Ser
University of the Basque Country UPV/EHU & Basque Center for Applied Mathematics (BCAM)
E-mail: javier.delser@ehu.eus, jdelser@bcamath.org

Eneko Osaba
TECNALIA, Basque Research and Technology Alliance (BRTA),
E-mail: eneko.osaba@tecnalia.com

Albert Bifet
ParisTech & The University of Waikato
E-mail: albert.bifet@telecom-paristech.fr, abifet@waikato.ac.nz

Francisco Herrera
Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI),
University of Granada
E-mail: herrera@decsai.ugr.es

Keywords Concept drift · Drift detection · Data stream mining · Cellular automata

1 Introduction

Data Stream Mining (DSM) techniques are focused on extracting patterns from continuous (potentially infinite) and fast data. A data stream is the basis of machine learning techniques for this particular kind of data, which is composed of an ordered sequence of instances that arrive one by one or in batches. Depending on the constraints imposed by the application scenario at hand, such instances can be read only once or at most a reduced number of times, using limited computing and memory resources. These constraints require an incremental learning (or one-pass learning) procedure where past data cannot be stored for batch training in future time steps. Due to these challenging conditions under which learning must be done, DSM has acquired a notable relevance in recent years, mostly propelled by the advent of Big Data technologies and data-intensive practical use cases [3].

In this context, data streams are often generated by non-stationary phenomena, which may provoke a change in the distribution of the data instances (and/or their annotation). This phenomenon is often referred to as *concept drift* [40]. These changes cause that predictive models trained over data streams become eventually obsolete, not adapting suitably to the new distribution (concept). The complexity of overcoming this issue, and its prevalence over many real scenarios, make *concept drift* detection and adaptation acknowledged challenges in DSM [28]. Examples of data stream sources undergoing *concept drift* include computer network traffic, wireless sensor data, phone conversations, social media, marketing data, ATM transactions, web searches, and electricity consumption traces, among others [38]. Recently, several emerging paradigms such as the so-called Smart Dust [21], Utility Fog [8], Microelectromechanical Systems (MEMS or "motes") [23], or Swarm Intelligence and Robotics [10], are in need for efficient and scalable solutions in real-time scenarios. Here *concept drift* may be present, and thus making drift detection a necessity.

This complexity in the *concept drift* phenomenon manifests when researchers try to characterize it [40]. Indeed, there are many different types of concept drifts, characterized by e.g. the speed or severity of change. Consequently, drift detection is a key factor for those active strategies that require triggering mechanisms for drift adaptation [20]. A drift detector estimates the time instants at which changes occur over the stream, so that when a change is detected, an adaptation mechanism is applied to the base learner so as to avoid a degradation of its predictive performance. The design of a *concept drift* detector with high performance is not trivial, yet it is crucial to achieve more reliable DSM models. In fact, a general-purpose strategy for *concept drift* detection, handling and recovery still remains as an open research avenue, as foretold by the fulfillment of the No Free Lunch theorem in this field [20]. This difficulty to achieve a universal best approach becomes evident in the most recent comparative among drift detectors made in [2]. Analyzing its mean rank of methods, we observe how there is no a method with the best metrics, or even showing the best performance in the majority of them. In this regard, the design objective is to develop techniques that detect all existing drifts in the stream with low latency and as few false alarms and missed detections

as possible. Thus, the most suitable drift detector depends on the characteristics of the DSM problem under study, giving more emphasis to some metrics than others. Regarding the detection metrics, we usually tend to put in value those drift detectors that are able to show a good classification performance while minimizing the distance of the true positive detections.

Cellular automata (CA), as low-bias and robust-to-noise pattern recognition methods with competitive classification performance, meet the requirements imposed by the aforementioned paradigms mainly due to their simplicity and parallel nature. In this work we present a Cellular aUtomaton for concept dRIft dEtection ($CURIE$), capable of competitively identifying drifts over data streams. The proposed approach is based on CA, which became popular when Conway's Game of Life appeared in 1970, and thereafter attracted attention when Stephen Wolfram published his CA study in 2002 [41]. Although CA are not very popular in the data mining field, Fawcett showed in [13] that they can become simple, low-bias methods. $CURIE$, as any other CA-based technique, is computationally complete (able to perform any computation which can be done by digital computers) and can model complex systems from simple structures, which puts it in value to be considered in the DSM field. Moreover, $CURIE$ is tractable and interpretable [25], both ingredients that have lately attracted attention under the eXplainable Artificial Intelligence (XAI) paradigm [1]. Next, we summarize the main contributions of $CURIE$ to the drift detection field:

- It is capable of competitively detecting abrupt and gradual concept drifts.
- It does not required the output (class prediction) of the base learner. Instead, it extracts the required information for drift detection from its internal structure, looking at the changes occurring in the neighborhood of cells.
- It is interpretable, due to the fact that its cellular structure is a direct representation of the feature space and the labels to be predicted.
- It can be combined with any base learner.

Besides, $CURIE$ offers other additional advantage in DSM:

- It is also able to act as an incremental learner and adapt to the change [25], going one step further by embodying an *all-in-one* approach (learner and detector).

The rest of the manuscript is organized as follows: first, we provide the background of the field in Sect. 2. Next, we introduce the fundamentals of CA and their role in DSM in Sect. 3. Sect. 4 exposes the details of our proposed drift detector $CURIE$, whereas Sections 5 and 6 elaborate on experimental setup and analyze results with synthetic and real-world data stream respectively. Finally, Sect. 7 concludes the manuscript with an outlook towards future research derived from this work.

2 Related Work

We now delve into the background literature related to the main topics of this work: drift detection (Subsection 2.1) and cellular automata for machine learning (Subsection 2.2).

2.1 Drift Detection

DSM has attracted much attention from the machine learning community [16]. Researchers are now on the verge of moving out DSM methods from laboratory environments to real scenarios and applications, similarly to what occurred with traditional machine learning methods in the past. Most efforts in DSM have been focused on supervised learning [3] (mainly on classification), addressing the *concept drift* problem [40]. Generally, these efforts have been invested in the development of new methods and algorithms that maintain an accurate decision model with the capability of learning incrementally from data streams while forgetting concepts [27,39].

For this purpose, drift detection and adaptation mechanisms are needed [28]. In contrast to passive (blind) approaches where the model is continuously updated every time new data instances are received (i.e., drift detection is not required), active strategies (where the model gets updated only when a drift is detected) are in need for effective drift detection mechanisms. Most active approaches usually utilize a specific classifier (base learner) and analyze its classification performance (e.g. accuracy or error rate) to indicate whether a drift has occurred or not. Then, the base learner is trained on the current instance within an incremental learning process repeated for each incoming instance of the data stream. Despite the most used input for the drift detectors are the accuracy or error rate, we can find other detectors that use other inputs such as diversity [29] or structural changes stemming from the model itself [26].

There is a large number of drift detectors in the literature, many of them compared in [17]. As previously mentioned, the conclusion of these and other works is that there is no a general-purpose strategy for *concept drift*. The selection of a good strategy depends on the type of drift and particularities of each data streaming scenario. Other more recent *concept drift* detection mechanisms have been presented and well described in [2].

2.2 Cellular Automata for Pattern Recognition

CA are not popular in the pattern recognition community, but even so we can find recent studies and applications. In [7], authors propose CA to simulate potential future impacts of climate change on snow covered areas, whereas in [18] an approach to explore future land use/cover change under different socio-economic realities and scales is presented. Scheduling is another field where CA has been profusely in use [5]. Another recent CA approach for classification is [36]. CA have been also used with convolutional neural networks [15] and reservoir computing [32].

Regarding DSM or *concept drift* detection fields, the presence of CA-based proposals is even scarcer. Although a series of contributions unveiled solid foundations for CA to be used for pattern recognition [22,34,6], it was not until 2008 [13] (departing from the seminal work in [35]) when CA was presented as a simple but competitive method for parallelism, with a low-bias, effective and competitive in terms of classification performance, and robust to noise. Regarding DSM and *concept drift* detection, timid efforts have been reported so far in [19] and [33], which must be considered as early attempts to deal with noise rather than

with incremental learning and drift detection. They used a CA-based approach as a real-time instance selector to avoid noisy instances, while the classification task was performed in batch learning mode by non-CA-based learning algorithms. Thus, CA is proposed as a mere complement to select instances, and not as an incremental learner. Besides, their detection approach is simply based on the local class disagreements between neighboring cells, without considering relevant aspects such as the grid size, the radius of the neighborhood, or the moment of the disagreement, among other factors. Above all, they do not provide any evidence on how their solution learns incrementally, nor details on the real operation of the drift detection approach. Finally, in terms of drift detection evaluation, their approach is not compared to known detectors using reputed base learners and standard detection metrics.

More recently, the authors of [25] transform a cellular automaton into a real incremental learner with drift adaptation capabilities. In this work, we go one step further by proposing $CURIE$, a cellular automaton featuring a set of novel ingredients that endow it with abilities for drift detection in DSM. As we will present in detail, $CURIE$ is an interpretable CA-based drift detector, able to detect abrupt and gradual drifts, and providing very competitive classification performances and detection metrics.

3 Cellular Automata

3.1 Foundations

Von Neumann described CA as discrete dynamical systems with a capacity of universal computability [37]. Their simple local interaction and computation of cells result in a huge complex behavior when these cells act together, being able to describe complex systems in several scientific disciplines.

Following the notation of Kari in [24], a cellular automaton can be formally defined as: $A \doteq (d, \mathcal{S}, f_{\boxplus}, f_{\circlearrowleft})$, with d denoting the dimension, $\mathcal{S}$ a group of discrete states, $f_{\boxplus}(\cdot)$ a function that receives as input the coordinates of the cell and returns the neighbors of the cell to be utilized by the update rule, and $f_{\circlearrowleft}(\cdot)$ a function that updates the state of the cell at hand as per the states of its neighboring cells. Hence, for a radius $r = 1$ *von Neumann's* neighborhood defined over a $d = 2$-dimensional grid, the set of neighboring cells and state of the cell with coordinates $\mathbf{c} = [i, j]$ are given by:

$$
\begin{aligned}
&f_{\boxplus}([i,j]) = \{[i,j+1],[i-1,j],[i,j-1],[i+1,j]\},\\
&S(\mathbf{c}) = S([i,j])\\
&= f_{\circlearrowleft}(S([i,j+1]), S([i-1,j]), S([i,j-1]), S([i+1,j])),
\end{aligned}
$$

i.e., the vector of states $S([i, j])$ of the $[i, j]$ cell within the grid is updated according to the local rule $f_{\circlearrowleft}(\cdot)$ when applied over its neighbors given by $f_{\boxplus}([i, j])$ (Figure 1). For a d-dimensional space, a *von Neumann's* neighborhood contains $2d$ cells.

A cellular automaton should present these three properties: i) *parallelism* or *synchronicity* (all of the updates to the cells compounding the grid are performed at the same time); ii) *locality* (when a cell $[i, j]$ is updated, its state $S[i, j]$ is based on the previous state of the cell and those of its nearest neighbors); and

iii) *homogeneity* or *properties-uniformity* (the same update rule $f_{\circlearrowleft}(\cdot)$ is applied to each cell).

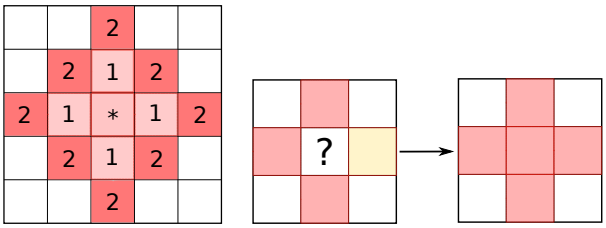


Fig. 1 Neighborhood of CA in data mining: (a) a von Neumann's neighborhood with radius $r = 1$ and $r = 2$ using the Manhattan distance; (b) the center cell inspects its *von Neumann's* neighborhood ($r = 1$) and applies the majority voting rule in a one-step update.

3.2 Cellular Automata for Data Stream Mining

A DSM process that may evolve over time can be defined as follows: given a time period $[0, t]$, the historical set of instances can be denoted as $\mathbf{D}_{0,t} = \mathbf{d}_0, \ldots, \mathbf{d}_t$, where $\mathbf{d}_i = (\mathbf{X}_i, y_i)$ is an instance, $\mathbf{X}_i$ is the vector of features, and y_i its label. Assuming that $\mathbf{D}_{0,t}$ follows a certain joint probability distribution $P_t(\mathbf{X}, y)$. As it has already been mentioned, data streams usually suffer from *concept drift*, which may change their data distribution, provoking that predictive models trained over them become obsolete. Thus, *concept drift* happens at timestamp $t + 1$ when $P_t(\mathbf{X}, y) \neq P_{t+1}(\mathbf{X}, y)$, i.e. as a change of the joint probability distribution of $\mathbf{X}$ and y at time t.

In addition to the presence of *concept drift*, DSM also imposes by itself its own restrictions, which calls for a redefinition of the previous CA for data mining. Algorithms learning from data streams must operate under a set of restrictions [12]:

- Each instance of the data stream must be processed only once.
- The time to process each instance must be low.
- Only a few data stream instances can be stored (limited memory).
- The trained model must provide a prediction at any time.
- The distribution of the data stream may evolve over time.

Therefore, when adapting a CA for DSM, the above restrictions must be taken into account to yield a CA capable of learning incrementally, and with drift detection and adaptation mechanisms. In order to use CA in DSM, data instances flowing over time must be mapped incrementally to the cells of the grid. Next, we analyze each of the mutually interdependent parts in CA for DSM:

- `Grid`: In a data mining problem with n features, the standard procedure adopted in the literature consists of assigning one grid dimension to each feature. After that, it is necessary to split each grid dimension by the values of the features, in a way that we obtain the same number of cells per dimension. To achieve

that, "bins" must be created for every dimension (Figure 2) by arranging evenly spaced intervals based on the maximum and minimum values of the features. These "bins" delimit the boundaries of the cells in the grid.
- `States:` We have to define a finite number of discrete states $|\mathcal{S}|$, which will correspond to the number of labels (classes) considered in the data mining problem.
- `Local rule:` In data mining the update rule $f(\cdot)$ can adopt several forms. One of the most accepted variants is a majority vote among neighbors' states (labels). For example, for $d = 2$:

$$S([i,j]) = \arg\max_{s \in \mathcal{S}} \sum_{[k,l] \in f_{\boxplus}([i,j])} \mathbb{I}(S([k,l]) = s),$$

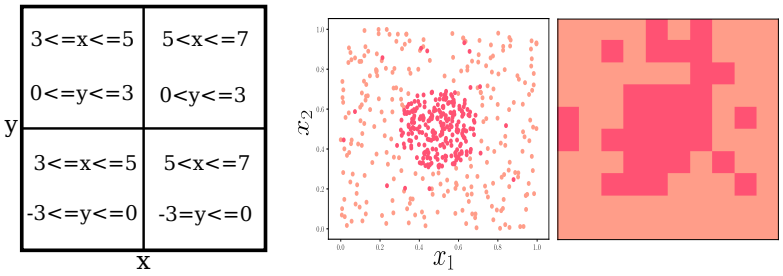
  where the value of $f_{\boxplus}([i,j])$ will be the coordinates of neighboring cells of $[i,j]$, and $\mathbb{I}(\cdot)$ is an auxiliary function taking value 0 when its argument is false and 1 if it is true.
- `Neighborhood:` a neighborhood and its radius must be specified. Even though a diversity of neighborhood relationships has been proposed in the related literature, the "von Neumann" (see Figure 1) or "Moore" are arguably the most resorted definitions of neighborhood for CA.
- `Initialization:` the grid is seeded with the feature values of the instances that belong to the training dataset. In order to decide the state of each cell, we assign the label corresponding to the majority of training data instances with feature values falling within the range covered by the cell. After that, cells of the grid are organized into regions of similar labels (Figure 2).
- `Generations:` when the initialization step finishes, some cells may remain unassigned, i.e. not all of them are assigned a state (label). In other words, the training dataset used to prepare the CA for online learning might not be large enough to "fill" all cells in the grid. In such a case, it becomes necessary to "evolve" the grid several times (generations) until all cells are assigned a state. In this evolving process, each cell calculates its new state by applying the update rule over the cells in its neighborhood. All cells apply the same update rule, being updated synchronously and at the same time. Here lies the most distinctive characteristic of CA: the update rule only inspects its neighboring cells, being the processing entirely local (Figure 1).

4 Proposed Approach: CURIE

We delve into the ingredients of *CURIE* to act as drift detector. As shown in Figure 3, its detection mechanism hinges on the evidence that a recent number of mutations in the neighborhood of a cell that has just mutated, may serve as an symptomatic indicator of the occurrence of a drift.

CURIE builds upon this intuition to efficiently identify drifts in data streams by fulfilling the following methodological steps (Algorithm 1):

- First, the CA inside *CURIE* is created by setting the value of its parameters (detailed as inputs in Algorithm 1), and following the characteristics of the given dataset (lines 1 to 5).



(a) (b)

Fig. 2 Data representation in CA: (a) a dataset with $d = 2$ dimensions, $|\mathcal{S}| = \{0, 1\}$, and $\mathcal{G} = 2$ "bins", where $\mathbf{X}_t = (X_t^1, X_t^2)$ falls between $[3, 7]$ (min/max X_t^1) and $[-3, -3]$ (min/max X_t^2); (b) data instances are mapped in the grid of a cellular automaton with $d = 2$ and $\mathcal{G} = 10$.

- A reduced number of *preparatory* instances of the data stream $[(\mathbf{X}_t, y_t)]_{t=0}^{P-1}$) is used to initialize the grid of $CURIE$. This grid is seeded with these instances, and then $CURIE$ is evolved for several iterations (generations) by applying the local rule until all cells are assigned a state i.e. the labels of the preparatory instances (lines 6 to 10).
- When the preparatory process is finished, we must ensure that several preparatory data instances have not seeded the same cell, because each cell must reflect only one single state. To this end, we must assign to each cell the most frequent state by inspecting the labels of all those instances that fell within its boundaries. Then, we must ensure that all cells have an assigned state by applying the local rule iteratively all over the grid. Since this last process can again seed a cell with several instances, we have to address this issue to ensure that the cell only reflects one single state (lines 11 to 13).
- Next, $CURIE$ starts predicting the data instances coming from the stream in a *test-then-train* fashion [14] (lines 14 to 28). This process consists of first predicting the label of the incoming instance, and next updating the limits of the cells in the grid should any feature value of the processed instance fall beyond the prevailing boundaries of the grid (lines 16 to 18). Secondly, the label of the incoming instance is used for training, i.e. for updating the state of the corresponding cell (line 19).
- In line 15 $CURIE$ stores the incoming instance in a sliding window W of size P, which is assumed, as in the related literature, to be small enough not to compromise the computational efficiency of the overall approach.
- During the *test-then-train* process, $CURIE$ checks if a *mutation* of the cell states has occurred (line 21). If the previous state of the cell (before the arrival of the incoming instance) is different from the label of the incoming instance, a mutation has happened. When there is a mutation, we assign the current time step to the cell in the grid of time steps (line 22). Then, $CURIE$ checks the state of the neighboring cells in a radius r_{mut} (of a von Neumann's neighborhood) in a specific period of time (line 23). If the number of neighboring mutants exceeds a threshold (line 24), $CURIE$ considers that a drift has occurred.
- After drift detection, it is time to adapt $CURIE$ to the detected change in the stream distribution. To this end, we reset the grid, the vector of states, and the vector of time steps in which a mutation was present (lines 25 to 27). Finally,

Algorithm 1: Steps of *CURIE* for drift detection and DSM

Input : Preparatory data instances $[(\mathbf{X}_t, y_t)]_{t=0}^{t=P-1}$; training/testing data for the rest of the stream $[(\mathbf{X}_t, y_t)]_{t=P}^{\infty}$; the grid size $\mathcal{G}$ ("bins" per dimension); a local update rule $f_{\circlearrowleft}(\cdot)$; neighborhood function $f_{\boxplus}(\mathbf{c})$ for cell with coordinates $\mathbf{c} \in \mathcal{G} = \{1, \ldots, G\}^d$; radius r for the neighborhood operator; radius r_{mut}; maximum number of allowed mutants *n_muts_allowed*; time period *mutation_period*; sliding window W of size P.

Output: Trained *CURIE* producing predictions $\widehat{y}_t$ $\forall t \in [P, \infty)$

1 Let d be equal to the number of features in $\mathbf{X}_t$
2 Let $|\mathcal{S}|$ be the number of classes (alphabet of y_t)
3 Set a vector of state hits per cell: $\mathbf{h_c} = []$ $\forall \mathbf{c} \in \mathcal{G}$, and a vector of mutations per time step and cell: $\mathbf{h_m} = []$ $\forall \mathbf{m} \in \mathcal{G}$
4 Initialize the limits of the grid: $[(lim_n^{low}, lim_n^{high})]_{n=1}^{d}$
5 Create the grid as per $\mathcal{G}$, n and $[(lim_n^{low}, lim_n^{high})]_{n=1}^{d}$
6 **for** $t = 0$ *to* $P - 1$ **do** `// Preparatory process`
7 Update limits as per $\mathbf{X}_t$, e.g., $lim_n^{low} = \min\{lim_n^{low}, x_t^n\}$
8 Update grid "bins" as per $\mathcal{G}$ and $[(lim_n^{low}, lim_n^{high})]_{n=1}^{d}$
9 Select the cell $\mathbf{c}$ in the grid that encloses $\mathbf{X}_t$
10 Append y_t to the vector of state hits $\mathbf{h_{c'}} = [\mathbf{h_{c'}}, y_t]$
11 Iterate with r and check $|\mathbf{h_c}|$ to ensure one state per cell in $\mathcal{G}$
12 Guarantee at least $|\mathbf{h_c}| = 1$ in all cells in $\mathcal{G}$
13 Iterate with r and recheck $|\mathbf{h_c}|$ to ensure one state per cell
14 **for** $t = P$ *to* ∞ **do** `// DSM processing`
15 Update W with the incoming instance $(\mathbf{X}_t, y_t)$
16 Predict $\widehat{y}_t$ as $S(\mathbf{c})$, with $\mathbf{c}$ denoting the coordinates of the cell enclosing $\mathbf{X}_t$
17 Update limits as per $\mathbf{X}_t$, e.g., $lim_n^{low} = \min\{lim_n^{low}, x_t^n\}$
18 Update "bins" as per $\mathcal{G}$ and $[(lim_n^{low}, lim_n^{high})]_{n=1}^{d}$
19 Save the current cell state: $cur\_st = S(\mathbf{c})$
20 Update $S(\mathbf{c}) = y_t$ (i.e. the verified class of test instance)
21 **if** $cur\_st \neq y_t$ **then** `// A mutation occurs in cell`
22 Append t to the vector of mutations: $\mathbf{h_{m'}} = [\mathbf{h_{m'}}, t]$
23 Calculate # mutant neighbors *n_muts* of the cell, within radius r_{mut} and within time *mutation_period*
24 **if** *n_muts* >= *n_muts_allowed* **then** `// Detection`
25 Initialize $\mathbf{h_m}, \mathbf{h_c}$
26 Initialize grid limits: $[(lim_n^{low}, lim_n^{high})]_{n=1}^{d}$
27 New grid as per $\mathcal{G}$, n, $[(lim_n^{low}, lim_n^{high})]_{n=1}^{d}$
28 Preparatory process $(6 - 10)$ with instances in W

the preparatory process is carried out by seeding the grid with the instances stored in the sliding window W (line 28).

Finally, after detailing the ingredients of *CURIE* to act as drift detector, we would like to highlight two improvements over [25] that positively impact on the learning of data distribution:

- If the predicted and the true label do not equal each other, the cell state in *CURIE* is always changed to the class of the incoming instance. Otherwise, if the age of the cell state (T_{age}) was considered, this could impact on drift detection resulting in more detection delay.
- In *CURIE* there is always one state assigned to each cell, thus it is not necessary to check the state of the closest cell among those with assigned state to provide a prediction. The cost of assigning one state to all cells of the grid is insubstantial; it is just carried out at the preparatory process and when drift is detected.

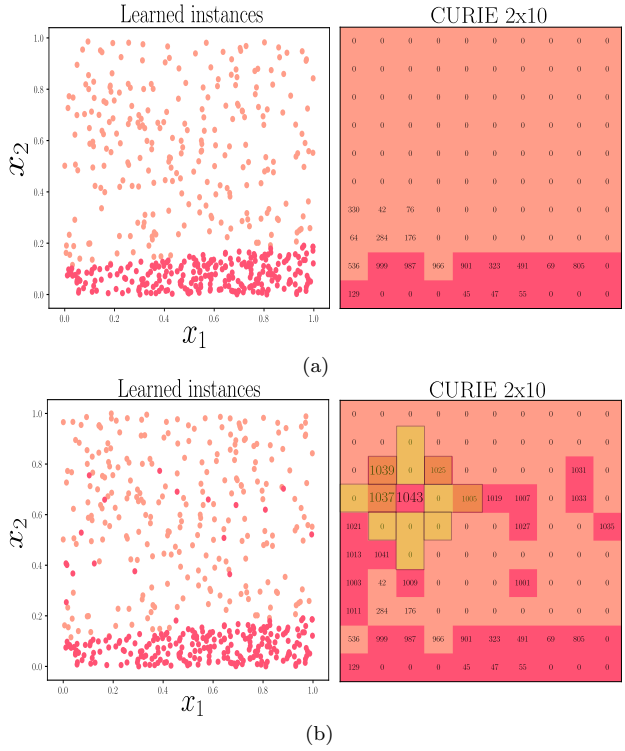


Fig. 3 The interpretable adaptation mechanism of $CURIE$ ($d \times \mathcal{G} = 2 \times 10$) based on the mutations of its neighborhood: (a) before the drift. $CURIE$ updates the time instants of each mutant cell, i.e. when the previous state of the cell (before the arriving of the incoming instance) is different from the label of the incoming instance itself; (b) drift occurs. $CURIE$ checks the neighborhood of each cell, and when at least 2 neighboring cells (defined by *n_muts_allowed* parameter) have mutated in the last 10 time steps (as per the *mutation_period* parameter), $CURIE$ considers that a drift has occurred. This is what is declared at $t = 1043$ with the cell $[2, 6]$ and its neighborhood of $r = 2$ (Manhattan distance), where 2 of its neighbors have mutated at time steps 1037 and 1039.

And so we achieve a more simple method that does not need to check the surroundings (neighborhood) of the cell when no state is assigned.

The source code of $CURIE$ is available at `https://github.com/TxusLopez/CURIE`.

5 Experimental Setup

In order to assess the performance of $CURIE$, we have designed several experiments with synthetic datasets configured with both abrupt and gradual drift versions.

Since drift detectors usually base their detection mechanisms on the prediction results of a base learner, both detection and classification are often set to work

together. As it has been already mentioned, $CURIE$ does not use the prediction of the base learner. Instead, it estimates the occurrence of the drift by looking at the changes that occur in the neighborhood of cells deployed over a grid that represents the distribution fo data. In our experiments we have accordingly combined three well-known base learners (`HT`, `NB` and `KNN`) with five popular drift detectors including our proposed detector (corr. `DDM`, `EDDM`, `PH`, `ADWIN`, and `CURIE`). They form 15 different learning-detection schemes following the algorithmic template shown in Algorithm 2. Such base learners and drift detection methods have been selected due to their wide use by the DSM community, and the availability of their implementations in the scikit-multiflow framework [30]. For more information, we refer the reader to [17] and [2]. Please note that the inclusion of `KNN` is not only based on its widely use, and it has also been considered due to its similarities with CA. While `KNN` is not strictly local (the neighborhood is not fixed beforehand and an the nearest neighbor of an instance may change), CA has a fixed neighborhood. In CA the local interaction between cells affects the evolution of each cell. We would also like to underline that the size of the sliding window of `KNN` (*max_window_size* parameter in Table 1) is the same than the number of recent instances that CA uses to be initialized and seeded after a drift is detected.

The computer used in the experiments is based on a $x86$ 64 architecture with 8 processors Intel(R) Core(TM) $i7$ at 2.70GHz, and $32DDR4$ memory running at $2,133$ MHz. The source code for the experiments is publicly available at this GitHub repository: `https://github.com/TxusLopez/CURIE`.

5.1 Datasets

In order to assess the performance of a drift detector by measuring the different detection metrics, we need to know beforehand where a real drift occurs. This is only possible with synthetic datasets. The scikit-multiflow framework [30] allows generating several kinds of synthetic data to simulate the occurrence of drifts. Concretely, we have generated 20 diverse synthetic datasets (10 abrupt and 10 gradual) by using several stream generators and functions, and with a different number of features and noise. They exhibit 4 concepts and 3 drifts at time steps $10,000$, $20,000$, and $30,000$ in the case of abrupt datasets, and at time steps $9,500$, $20,000$, and $30,500$ in the case of gradual ones. In the latter case, the width of the drift is $1,000$ time steps. All generated data streams have $40,000$ instances in total. Next, the details of the datasets:

- With `Sine` generator: it is ruled by a sequence of classification functions. `Sine_A` refers to abrupt cases and `Sine_G` to gradual ones. In the case of `Sine_F1`, the order of the functions is SINE1 ,reversed SINE1, SINE2, and reversed SINE2. For `Sine_F2` the order is namely, reversed SINE2-SINE2-reversed SINE1-SINE1. Therefore, `Sine` stream generator provides 4 different datasets: `Sine_A_F1`, `Sine_A_F2`, `Sine_G_F1`, and `Sine_G_F2`. They consist of 2 numerical features without noise, and a balanced binary class .
- With `Random Tree` generator: it is ruled by a sequence of tree random state functions. The parameters *max_tree_depth*, *min_leaf_depth*, and *fraction_leaves_per_level* were set to 6, 3, and 0.15 respectively. `RT_A` refers to abrupt cases and `RT_G` to gradual ones. In the case of `RT_F1`, the order of the functions is 8873-9856-7896-2563. For `RT_F2` the order is reversed 2563-7896-9856-8873. Therefore, `Random`

`Tree` stream generator provides 4 different datasets: `RT_A_F1`, `RT_A_F2`, `RT_G_F1`, and `RT_G_F2`. They consist of 2 numerical features without noise, and balanced binary class.
- With `Mixed` generator: it is ruled by a sequence of classification functions. `Mixed_A` refers to abrupt cases and `Mixed_G` to gradual ones. In the case of `Mixed_F1`, the order of the functions is 0-1-0-1. For `Mixed_F2` the order is reversed 1-0-1-0. Therefore, `Mixed` stream generator provides 4 different datasets: `Mixed_A_F1`, `Mixed_A_F2`, `Mixed_G_F1`, and `Mixed_G_F2`. They consist of 4 numerical features without noise, and a balanced binary class.
- With `Sea` generator: it is ruled by a sequence of classification functions. `Sea_A` refers to abrupt cases and `Sea_G` to gradual ones. In the case of `Sea_F1`, the order of the functions is 0-1-2-3. For `Sea_F2` the order is reversed 3-2-1-0. Therefore, `Sea` stream generator provides 4 different datasets: `Sea_A_F1`, `Sea_A_F2`, `Sea_G_F1`, and `Sea_G_F2`. They consist of 3 numerical features, a balanced binary class, and with the probability that noise will happen in the generation of 0.2 (probability range between 0 and 1).
- With `Stagger` generator: it is ruled by a sequence of classification functions. `Stagger_A` refers to abrupt cases and `Stagger_G` to gradual ones. In the case of `Stagger_F1`, the order of the functions is 0-1-2-0. For `Stagger_F2` the order is reversed 2-1-0-2. Therefore, `Stagger` stream generator provides 4 different datasets: `Stagger_A_F1`, `Stagger_A_F2`, `Stagger_G_F1`, and `Stagger_G_F2`. They consist of 3 numerical features without noise, and a balanced binary class.

Finally, as it is explained in section 3.2, it is necessary to create "bins" by splitting each grid dimension by the values of the features. For `Sine` and `RT` datasets we have used 20 "bins" per dimension, while for the rest of datasets we have used 10 "bins'. The values have been found experimentally, just knowing that a small grid is not capable of representing the data distribution (e.g. the grid of Figure 1). Here, we would like to warn other researches by underlining that `CURIE` exhibits at this moment a drawback that should be considered. Due to its exponential complexity, we recommend the use of `CURIE` in datasets with a low number of features. This setback can be tackled by carrying out the search over the grids cells by parallelizing this process.

The datasets are available at this Harvard Dataverse repository: `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/5OWRGB`.

5.2 Methods and Parameters

As for *CURIE* we have assigned one grid dimension to each feature of the dataset at hand. We also note that we have used a reduced number of instances to warm up the learning-detection schemes, and also *CURIE* (see Algorithms 1 and 2). The number of instances for this purpose usually depends on the memory or processing time restrictions. In our experiments we have considered a very low number of them in order to simulate a very restrictive real-time environment (see parameter P in Table 1). In all of them, *CURIE* has been configured with a *von Neumann's* neighborhood rather than opting for its *Moore* alternative. A *von Neumann's* neighborhood is linear in the number of dimensions of the instance space, and therefore scales well for problems of high dimensionality. In addition, a *Moore's* neighborhood includes more neighbors, thus we would have to potentially

apply the local rule over more cells. This would make the process computationally heavier and less suited for a DSM setting in the preparatory process and after the drift occurs.

The parameter configurations for the drift detectors under consideration are detailed in Table 1. The number of preparatory instances (P) and the sliding window (W) of size P are shared between $CURIE$ and the base learners. Concretely, their values are $P = 50$ and $w = P$. The values for the base learners parameters have been found through a hyper-parameter tuning process (Grid Search) carried out with these preparatory instances. Finally, Algorithm 2 presents the details of the learning and detection scheme followed by the experiments.

Table 1 Configuration of base learners and detectors.

Detector	Parameters	Value
DDM	$min\_num\_instances$	30
	α ($warning\_level$)	2.0
	β ($out\_control\_level$)	300
EDDM	$min\_num\_instances$	30
	α ($warning\_level$)	0.95
	β ($out\_control\_level$)	0.9
ADWIN	δ	0.002
PH	$min\_instances$	30
	δ	0.005
	$threshold$	50
	α	0.9999
CURIE	$f_{\boxplus}(\cdot)$	von Neumann
	$f_{\circlearrowleft}(\cdot)$	Majority voting
	r, r_{mut}	2,2
	$\lvert\mathcal{S}\rvert$	$\{0, 1\}$
	$d \times \mathcal{G}$	$n\_features \times n\_bins$
	$mutation\_period$	10
	$num\_mutants\_neighbors$	2

5.3 Performance Metrics

Regarding the classification accuracy, we have adopted the so-called *prequential accuracy* ($pACC$) [9], which is widely applied in streaming classification scenarios. This metric evaluates the base learner performance by quantifying the average accuracy obtained by the prediction of each test instance before its learning in an online *test-then-train* fashion. This accuracy metric can be defined as:

$$pACC(t) = \begin{cases} pACC_{ex}(t), \text{ if } t = t_{ref}; \text{ otherwise} \\ \\ preACC_{ex}(t-1) + \dfrac{pACC_{ex}(t) - pACC_{ex}(t-1)}{t - t_{ref} + 1}, \end{cases}$$

where $pACC_{ex}(t) = 0$ if the prediction of the test instance at time t before its learning is wrong, and 1 when it is correct. The reference time t_{ref} fixes the first

Algorithm 2: Learning-detection scheme

Input : Preparatory data instances $[(\mathbf{X}_t, y_t)]_{t=0}^{t=P-1}$; training/testing data for the rest of the stream $[(\mathbf{X}_t, y_t)]_{t=P}^{\infty}$; a sliding window W of size P

. **Output:** Trained base learners producing predictions $\widehat{y}_t$ $\forall t \in [P, \infty)$

Base learner ∈ [`HT`, `NB`, `KNN`]

Initialize base learners parameters of Table 1

Detector ∈ [`DDM`, `EDDM`, `ADWIN`,`PH`,`CURIE`]

Initialize detectors parameters of Table 1

for $t = 0$ *to* $P - 1$ **do** `// Preparatory process`

  if *detector* = *`CURIE`* **then**

    Train detector with $(\mathbf{X}_t, y_t)$

  Train base learner with $(\mathbf{X}_t, y_t)$

for $t = P$ *to* ∞ **do** `// DSM processing`

  Update W with the incoming instance $(\mathbf{X}_t, y_t)$

  Predict $\widehat{y}_t$

  Train base learner with $(\mathbf{X}_t, y_t)$

  if *detector* = *`CURIE`* **then**

    Train detector with $(\mathbf{X}_t, y_t)$

  else

    if $\widehat{y}_t \neq y_t$ **then**

      detector.add_element(0)

    else

      detector.add_element(1)

  if *detector.detected_change()* **then** `// Detection`

    Initialize detector

    Preparatory process $(6 - 7)$ with instances in W

Compare classification and detection performance metrics

time step used in the estimation, and allows isolating the computation of the prequential accuracy before and after a drift has occurred.

To know about the resources used by stream learners, we have adopted the measure RAM-Hours proposed in [4], based on rental cost options of cloud computing services. Here, 1 RAM-Hour equals 1 GB of RAM dispensed per hour of processing (GB-Hour). In order to analyze the *concept drift* identifications we have used several detection metrics based on true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN):

- `Precision:` defined as TP/(TP+FP), is the proportion of predicted drifts that are real drifts.
- `Recall:` defined as TP/(TP+FN), is the proportion of the real drifts that have been correctly detected.
- `Matthews correlation coefficient` (MCC): it is a correlation coefficient between the current and predicted instances. It returns values in the $[-1, 1]$ range. It is defined as:

$$MCC = \frac{((TP \cdot TN) - (FP \cdot FN))}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}.$$

We have also measured the distance of the drift detection to the real drift occurrence (μD). Finally, it is worth mentioning that the drifts detected within 2% and 10% (for abrupt and gradual drifts respectively) of the concept size after the real drift positions were computed as TP.

5.4 Statistical Tests

We have statistically compared the detectors in all datasets by carrying out the Friedman non-parametric statistical test as described in [11]. This test is the first step to know whether any of the detectors have a performance statistically different (in prequential accuracy, RAM-Hours, µD, and MCC) from the others. The null hypothesis states that all detectors are statistically equal, and in all cases was rejected. Then it is necessary to use a post-hoc test to discover in what detectors there is a statistical difference (in prequential accuracy, RAM-Hours, µD, and MCC), and we used the Nemenyi post-hoc test [31] with 95% confidence to compare all the detectors against all the others. The results are graphically presented showing the critical difference (CD) represented by bars and detectors connected by a bar are not statistically different.

6 Results and Discussion

For the sake of space, we only present the mean results in Table 2. Detailed results are provided in `https://github.com/TxusLopez/CURIE`.

Table 2 Mean results and mean ranks of the detectors in each metric for all considered datasets. $pACC$ compiles the prequential accuracy results of those base learners (`HT`, `KNN`, and `NB`) which have been hybridized with each detector (`DDM`, `EDDM`, `ADWIN`, `PH`, and `CURIE`). $RAM-Hours$ provides the costs of each mentioned hybrid, while µD and MCC show the results for the detection metrics. Note that µD equals 1000 when there are no true positives; otherwise, if µD would equal i.e. 0, we would favor this metric.

		DDM	EDDM	ADWIN	PH	CURIE
pACC	**score**	0.81	0.79	0.83	0.81	0.83
	rank	2.72	4.00	2.18	3.24	2.81
RAM-Hours	**score**	0.0005448	0.0007455	0.0005708	0.0004462	0.0009449
	rank	3.31	2.56	3.00	2.32	3.82
µD	**score**	1000.00	698.80	594.59	892.68	465.45
	rank	3.81	2.85	2.40	3.54	1.90
MCC	**score**	0.00	0.06	0.26	0.06	0.37
	rank	3.93	3.22	2.53	3.56	1.76

Figure 4 presents the evaluation of the concept drift detection methods based on the results of the Table 2.

In Table 2 we observe that `CURIE` and `ADWIN` achieve the best $pACC$ metric with 0.83. However, `CURIE` is the worst in terms of $RAM-Hours$ with 0.0009449. Here, in favor of `CURIE`, it is worth mentioning that it is competing with well-established detectors whose code have been optimized and tested by the community in the scikit-multiflow framework [30]. Probably, future versions of `CURIE` will be more competitive in terms of this metric. Regarding detection metrics, `CURIE` is the best with 465.45 for µD and 0.37 for MCC.

According to the ranks of Figure 4, `ADWIN`, `DDM` and `CURIE` are the best detectors in terms of $pACC$, yet no statistical differences between them. Regarding the $RAM-Hours$ metric, `CURIE` and `DDM` are the worst detectors, with no statistical differences between them. However, in what refers to µD and MCC, `CURIE`, `ADWIN`, and `EDDM` are the best detectors, yet again no statistical differences between them.

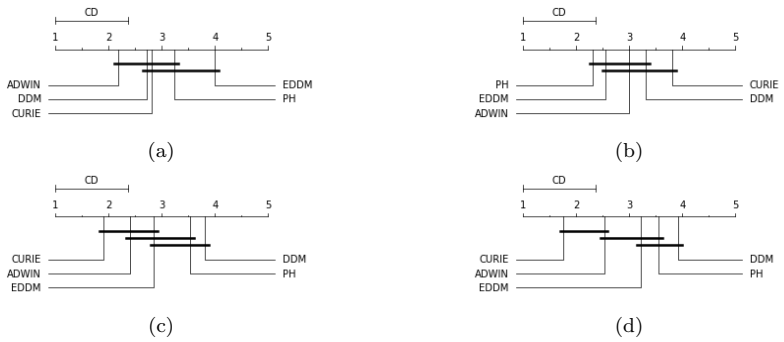


Fig. 4 Comparison of (a) **pACC**, (b) **RAM-Hours**, (c) **µD**, and (d) **MCC** results of detectors using the Nemenyi test based on the results of Table 2 with a 95% confidence interval. CD is 1.363887 for 5 detectors and 20 datasets.

The conditions of the Nemeneyi test have been very tight (95% confidence for 5 detectors in 20 datasets) and it is difficult to achieve statistical differences. Even so, `CURIE` has shown to be an interpretable drift detector competitive in terms of predictive performance ($pACC$) and detection metrics (µD and MCC), without depending on the output (class prediction) of the base learner. Moreover, `CURIE` provides competitive metrics for abrupt and gradual drifts, being this issue very controversial in the drift detection field, as it was shown in [17].

7 Conclusion and Outlook

This work has presented $CURIE$, a competitive and interpretable drift detector based on cellular automata. Until now, cellular automata have shown to be suitable solutions for data mining tasks due to their simplicity to model complex systems, being robust to noise and a low-bias method. Besides, they are computationally complete with parallelism capacity, and they already showed competitive classification performances as data mining methods.

This time, we have focused on their capacity to detect *concept drift*. They have revealed themselves as suitable detectors that achieve competitive detection metrics. They have also allowed base learners to exhibit competitive classification accuracies in a diversity of datasets subject to abrupt and gradual concept drifts. They are suitable candidates to represent data distributions with a few instances, being this ability welcomed in data stream mining tasks where memory and computational resources are often severely constrained. Moreover, $CURIE$ can act as an all-in-one approach, in contrast to many other drift detectors which are based on a combination of a base learner method with a detection mechanism.

As future work, we aim to extend the experimental benchmark to more synthetic and real datasets in order to extrapolate the findings and conclusions of this study to different types of drift and more realistic applications. Applying ensemble approaches or even networks of cellular automata are also among our subjects of further study.

Acknowledgements This work has received funding support from the ECSEL Joint Undertaking (JU) under grant agreement No 783163 (*iDev40* project). The JU receives support from the European Union's Horizon 2020 research and innovation programme, national grants from Austria, Belgium, Germany, Italy, Spain and Romania, as well as the European Structural and Investment Funds. Authors would like to also thank the ELKARTEK and EMAITEK funding programmes of the Basque Government (Spain).

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.
2. Roberto Souto Maior Barros and Silas Garrido T Carvalho Santos. A large-scale comparison of concept drift detectors. *Information Sciences*, 451:348–370, 2018.
3. Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press, 2018. `https://moa.cms.waikato.ac.nz/book/`.
4. Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank. Fast perceptron decision tree learning from evolving data streams. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 299–310. Springer, 2010.
5. Tiago I Carvalho, Murillo G Carneiro, and Gina MB Oliveira. Improving cellular automata scheduling through dynamics control. *International Journal of Parallel, Emergent and Distributed Systems*, 34(1):115–141, 2019.
6. Parimal Pal Chaudhuri, Dipanwita Roy Chowdhury, Sukumar Nandi, and Santanu Chattopadhyay. *Additive cellular automata: theory and applications*, volume 43. John Wiley & Sons, 1997.
7. Antonio-Juan Collados-Lara, Eulogio Pardo-Igúzquiza, and David Pulido-Velazquez. A distributed cellular automata model to simulate potential future impacts of climate change on snow cover area. *Advances in water resources*, 124:106–119, 2019.
8. Amir Vahid Dastjerdi and Rajkumar Buyya. Fog computing: Helping the internet of things realize its potential. *Computer*, 49(8):112–116, 2016.
9. A Philip Dawid, Vladimir G Vovk, et al. Prequential probability: Principles and properties. *Bernoulli*, 5(1):125–162, 1999.
10. Javier Del Ser, Eneko Osaba, Daniel Molina, Xin-She Yang, Sancho Salcedo-Sanz, David Camacho, Swagatam Das, Ponnuthurai N Suganthan, Carlos A Coello Coello, and Francisco Herrera. Bio-inspired computation: Where we stand and what's next. *Swarm and Evolutionary Computation*, 48:220–250, 2019.
11. Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.
12. Pedro Domingos and Geoff Hulten. A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949, 2003.
13. Tom Fawcett. Data mining with cellular automata. *ACM SIGKDD Explorations Newsletter*, 10(1):32–39, 2008.
14. João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.
15. William Gilpin. Cellular automata as convolutional neural networks. *Physical Review E*, 100(3):032402, 2019.
16. Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and Joao Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter*, 21(2):6–22, 2019.

17. Paulo M Gonçalves Jr, Silas GT de Carvalho Santos, Roberto SM Barros, and Davi CL Vieira. A comparative study on concept drift detectors. *Expert Systems with Applications*, 41(18):8144–8156, 2014.
18. Dimitrios Gounaridis, Ioannis Chorianopoulos, Elias Symeonakis, and Sotirios Koukoulas. A random forest-cellular automata modelling approach to explore future land use/cover change in attica (greece), under different socio-economic realities and scales. *Science of the Total Environment*, 646:320–335, 2019.
19. Sattar Hashemi, Ying Yang, Majid Pourkashani, and Mohammadreza Kangavari. To better handle concept change and noise: a cellular automata approach to data stream classification. In *Australasian Joint Conference on Artificial Intelligence*, pages 669–674. Springer, 2007.
20. Hanqing Hu, Mehmed Kantardzic, and Tegjyot S Sethi. No free lunch theorem for concept drift detection in streaming data classification: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page e1327, 2019.
21. Mohammad Ilyas and Imad Mahgoub. *Smart Dust: Sensor network applications, architecture and design*. CRC press, 2018.
22. Erica Jen. Invariant strings and pattern-recognizing properties of one-dimensional cellular automata. *Journal of statistical physics*, 43(1-2):243–265, 1986.
23. Jack W Judy. Microelectromechanical systems (mems): fabrication, design and applications. *Smart materials and Structures*, 10(6):1115, 2001.
24. Jarkko Kari. Theory of cellular automata: A survey. *Theoretical computer science*, 334(1-3):3–33, 2005.
25. Jesus L. Lobo, Javier Del Ser, and Francisco Herrera. Lunar: Cellular automata for drifting data streams. *Information Sciences*, 2020.
26. Jesus L Lobo, Javier Del Ser, Ibai Laña, Miren Nekane Bilbao, and Nikola Kasabov. Drift detection over non-stationary data streams using evolving spiking neural networks. In *International symposium on intelligent and distributed computing*, pages 82–94. Springer, 2018.
27. Viktor Losing, Barbara Hammer, and Heiko Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, 2018.
28. Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 2018.
29. Leandro L Minku and Xin Yao. Ddd: A new ensemble approach for dealing with concept drift. *IEEE transactions on knowledge and data engineering*, 24(4):619–633, 2011.
30. Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. Scikit-multiflow: a multi-output streaming framework. *The Journal of Machine Learning Research*, 19(1):2915–2914, 2018.
31. Peter Nemenyi. Distribution-free multiple comparisons. In *Biometrics*, volume 18, page 263. International Biometric Soc 1441 I ST, NW, SUITE 700, WASHINGTON, DC 20005-2210, 1962.
32. Stefano Nichele and Andreas Molund. Deep learning with cellular automaton-based reservoir computing. *Complex Systems*, 2017.
33. Majid Pourkashani and Mohammad Reza Kangavari. A cellular automata approach to detecting concept drift and dealing with noise. In *2008 IEEE/ACS International Conference on Computer Systems and Applications*, pages 142–148. IEEE, 2008.
34. Raghu Raghavan. Cellular automata in pattern recognition. *Information Sciences*, 70(1-2):145–177, 1993.
35. Alfred Ultsch. Data mining as an application for artificial life. In *Proc. Fifth German Workshop on Artificial Life*, pages 191–197. Citeseer, 2002.
36. Arif Orhun Uzun, Tuğba Usta, Enes Burak Dündar, and Emin Erkan Korkmaz. A solution to the classification problem with cellular automata. *Pattern Recognition Letters*, 116:114–120, 2018.
37. John Von Neumann, Arthur W Burks, et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1966.
38. Indrè Žliobaitè, Mykola Pechenizkiy, and Joao Gama. An overview of concept drift applications. In *Big data analysis: new algorithms for a new society*, pages 91–114. Springer, 2016.
39. Scott Wares, John Isaacs, and Eyad Elyan. Data stream mining: methods and challenges for handling concept drift. *SN Applied Sciences*, 1(11):1412, 2019.

40. Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.
41. Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, IL, 2002.